FungAI

Classificatore di funghi Velenosi e Commestibili

Festa Francesco 0512113547

Progetto per il corso di Machine Learning
Anno Accademico 2023/2024

Sommario

1. Introduzione **2**
2. Data Understanding and Preparation **3**
2.1. Data Exploration 3
2.2. Missing Values 4
2.3. Data Distributions 5
2.4. Feature Encoding 6
2.5. Data Splitting 6
3. Data Modeling **7**
4. Evaluation **8**
4.1. Naive Bayes 9
4.2. Decision Tree 10
4.3. Random Forest 11
4.4. K-Nearest Neighbors 12
4.5. Support Vector Machines 13
4.6. Comparisons 14
4.7. Decision Tree Graph 15
5. Deployment **16**
5.1. Web Application 16
5.2. Explainability 17
6. Conclusions **18**

1. Introduzione

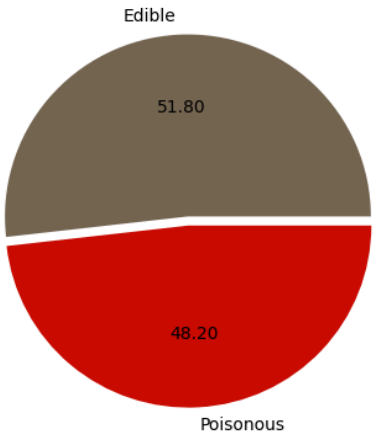
L’ obiettivo di questo progetto è quello di sviluppare un agente intelligente che possa classificare correttamente i funghi velenosi da quelli commestibili fornendo una descrizione delle loro caratteristiche visive e olfattive. Per affrontare il problema si è partiti utilizzando il seguente dataset (https://archive.ics.uci.edu/dataset/73/mushroom) fornito da UC Irvine. In particolare esso contiene 22 feature categoriche e per ogni riga la corrispettiva label (Velenoso o Commestibile). Per garantire delle buone prestazioni verranno implementati più classificatori, così da poter confrontare le metriche di valutazione di ognuno (Con una particolare attenzione alla Recall, in quanto all’interno del dominio di questo problema preferiamo dare un falso positivo, quindi un fungo commestibile classificato come velenoso, rispetto a un falso negativo, un fungo velenoso classificato come commestibile).
Tutto il codice e i grafici prodotti (Anche quelli non presenti all’interno di questo documento) sono reperibili alla seguente repository GitHub: https://github.com/MonTheDog/FungAI.
Inoltre è possibile testare il modello prodotto al seguente link: https://fungai.streamlit.app/.

2. Data Understanding and Preparation

Partiamo dal lavorare sui dati del nostro dataset, per trovare e sistemare eventuali problemi che potrebbero impattare negativamente le prestazioni del nostro modello

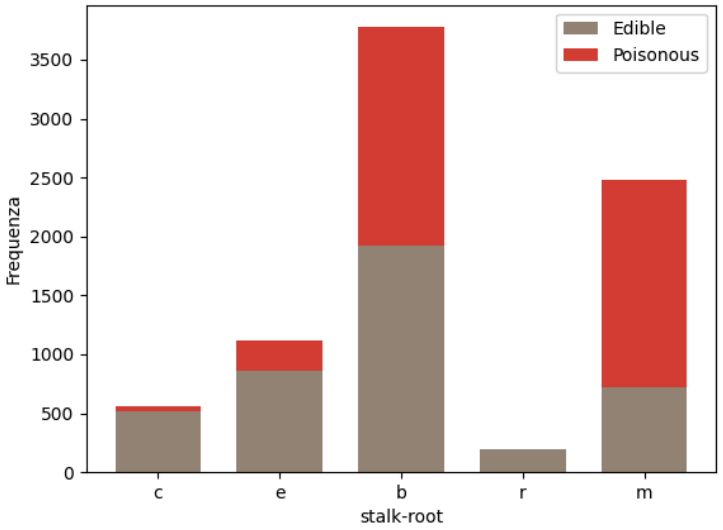
2.1. Data Exploration

Da una primissima analisi del dataset scopriamo subito che contiene 8124 righe e 23 colonne (22 feature e 1 target, in linea con quanto documentato nella repository). Salta subito all'occhio però che nel caricamento le variabili sono divenute tutte di tipo "object" essendo caratteri (E quindi variabili categoriche). Questo potrebbe causare problemi in un secondo momento, in particolare nella produzione dei grafici, andiamo quindi a convertirle tutte in tipo "string".
Passiamo ora a controllare se ci sono valori mancanti. Sia la documentazione che l'analisi effettuata riportano esclusivamente 2480 valori mancanti nella colonna "stalk-root". Vedremo dopo come è stato gestito questo problema.
Sono stati poi controllati i valori duplicati all'interno del dataset, ma fortunatamente in questo caso non ne abbiamo, quindi possiamo procedere senza problemi.
Infine è stato fatto un controllo sul bilanciamento del dataset, che ci ha riportato che il 51.8% delle righe ha la label "Commestibile", mentre il restante 48.2% ha la label "Velenoso". Questo significa che il dataset è già molto bilanciato di per sé, quindi possiamo non effettuare operazioni a tal merito.



2.2. Missing Values

Andiamo ora ad affrontare i valori mancanti nella colonna “Stalk-root”. Iniziamo col fare un’analisi più approfondita della distribuzione di questi valori sulle due classi, così da comprendere quanto questa feature possa effettivamente tornarci utile. (Con “m” sono denotati i valori mancanti).

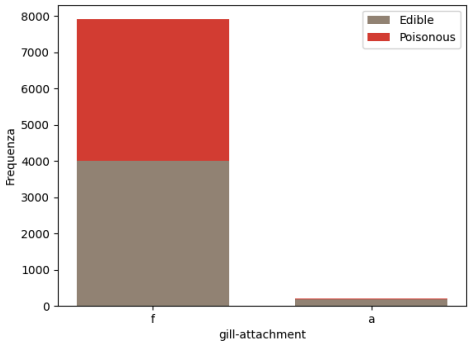


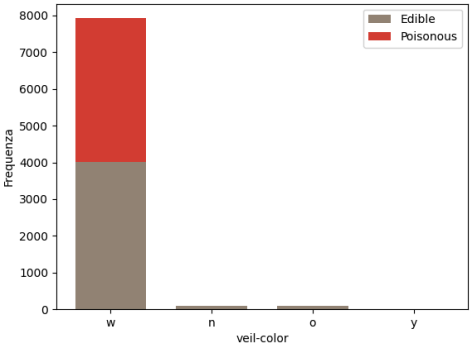
Notiamo immediatamente che “c” ed “r” hanno una discreta potenza predittiva, mentre“b” ed “e” hanno una distribuzione più varia, così come il valore mancante. Questo fa pensare che sarebbe un peccato perdere le feature con buona potenza predittiva per rimuoverne una che probabilmente andrà ad influire poco nelle altre previsioni. Per questo motivo si è deciso di sostituire tutti i valori mancanti con un valore arbitrario “m”, aggiungendo quindi una nuova caratteristica all’interno del dominio di questa feature.

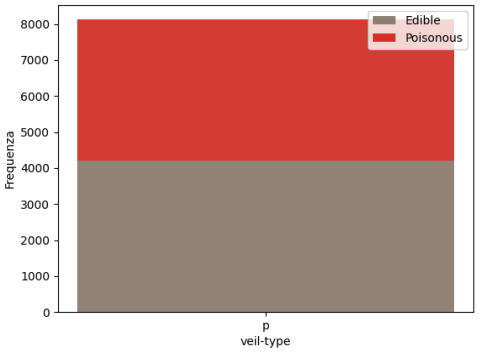
2.3. Data Distributions

Sono stati poi prodotti gli istogrammi di ogni feature, per cercare correlazioni interessanti tra queste e la variabile target. Da questo processo, sono state ricavate una serie di osservazioni:

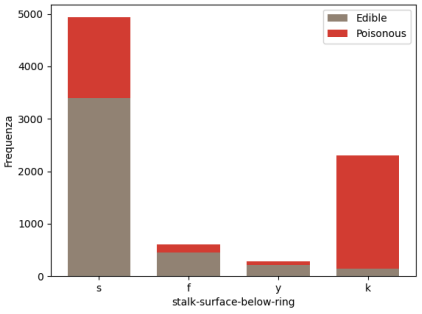
1. Le colonne “Gill-attachment” e “Veil-color” hanno una bassissima varianza, mentre “Veil-type” ha una varianza pari a 0. Per questo motivo tutte e tre le feature sono state rimosse, in quanto possedevano pochissima potenza predittiva, così da velocizzare il modello.

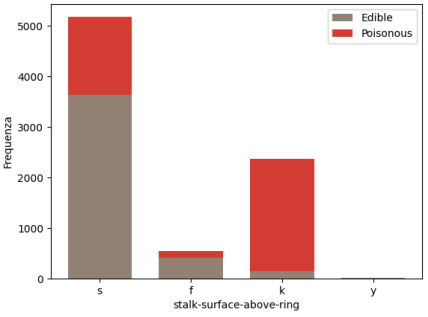


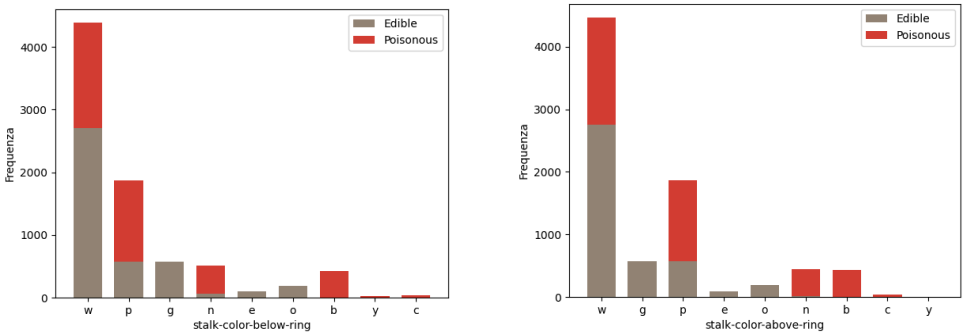




2. Le coppie di colonne “Stalk-surface-above ring”, “Stalk-surface-below-ring” e “Stalk-color-above-ring”, “Stalk-color-below-ring” sono estremamente simili, in quanto rappresentano due feature del fungo che nella maggior parte dei casi corrispondono. Per questo motivo si è deciso, per ogni coppia, di rimuoverne una e rinominare l’altra così da velocizzare il modello.







3. I domini delle singole feature non sono gli stessi riportati dalla documentazione. Infatti sulla repository sono riportati alcuni valori che non compaiono mai nel dataset. Questa informazione di per sé non è particolarmente importante in questa fase, ma lo sarà in fase di Deployment, quando il modello dovrà essere utilizzato dall'utente finale. Per questo motivo i valori che non appaiono mai non saranno inseribili, riducendo gli input a solo quelli che effettivamente compaiono nel dataset.

2.4. Feature Encoding

Ora che abbiamo le nostre feature di training, dobbiamo affrontare il problema delle feature categoriche. Difatti i modelli di Machine Learning non lavorano bene con queste, ma devono lavorare con feature numeriche. Un approccio naïve potrebbe essere usare un Label Encoder, che associa ad ogni valore possibile della variabile un numero, ma questo approccio nel nostro problema è sbagliato: infatti così facendo si verrebbe a creare una correlazione tra dati non correlati, poiché il modello potrebbe interpretare un numero alto come migliore, quando in realtà i numeri non hanno alcun significato intrinseco, ma sono stati usati solo per codificare le diverse categorie. Per questo motivo un approccio molto più corretto è il One Hot Encoding, che va a dividere la feature in tante piccole feature booleane quanti sono i possibili valori categorici, per poi assegnare 0 o 1 in base al valore effettivo. Così facendo avremo molte più feature, ma permetteremo al modello di fare previsioni senza alcun bias dato da un'errata codifica.

2.5. Data Splitting

Arrivati a questo punto, il dataset è pronto per essere usato per il training dei modelli. Non ci resta che dividerlo in due dataset più piccoli, di cui uno verrà utilizzato per il training (80% del dataset) e l'altro per il testing del modello (20% del dataset). Questo viene fatto per evitare di testare il modello su dati che ha già visto in fase di training, così da evitare il rischio che abbia "imparato a memoria" il dataset e testare l'effettiva qualità delle inferenze.

3. Data Modeling

In questa fase sono stati addestrati cinque diversi classificatori, così da poter poi confrontare le prestazioni di ognuno di essi e selezionare il migliore. I classificatori selezionati sono:

- Naive Bayes
- Decision Tree
- Random Forest
- K-Nearest Neighbors
- Support Vector Machines

Una volta effettuato l'addestramento possiamo procedere alla fase di Valutazione.

4. Evaluation

Per ognuno dei modelli summenzionati, verranno utilizzate le seguenti metriche di valutazione:

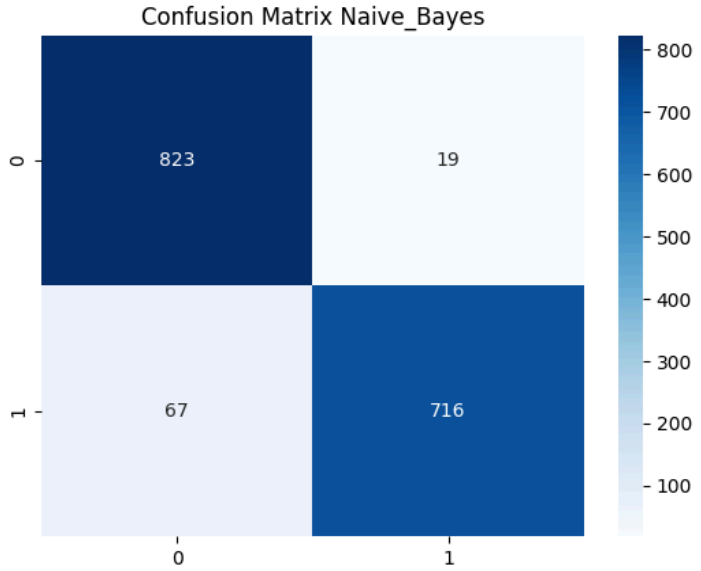
- Accuracy
- Precision
- Recall
- F1
- AUC

Inoltre per ciascuno verrà mostrata:

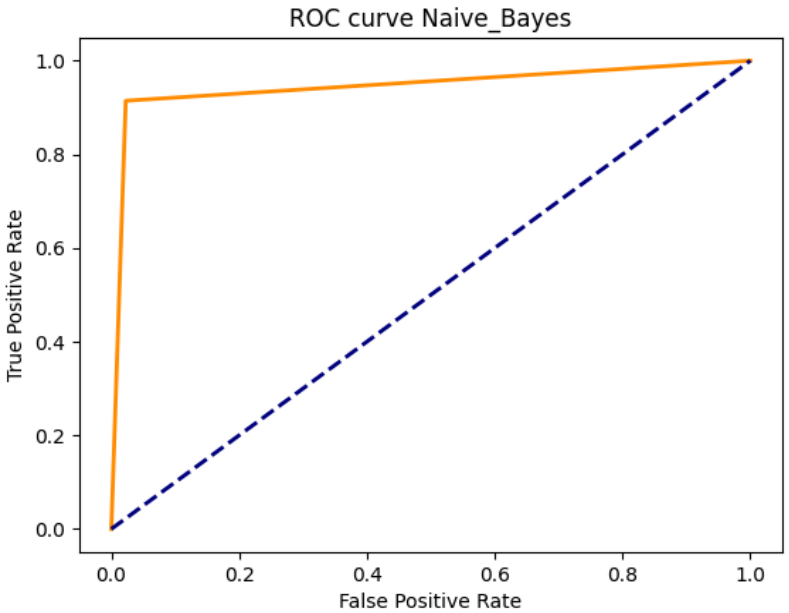
- Confusion Matrix
- ROC Curve

4.1. Naive Bayes

- Accuracy: 94.71%
- Precision: 97.41%
- Recall: 91.44%
- F1: 94.33%
- AUC: 0.9459
- Confusion Matrix

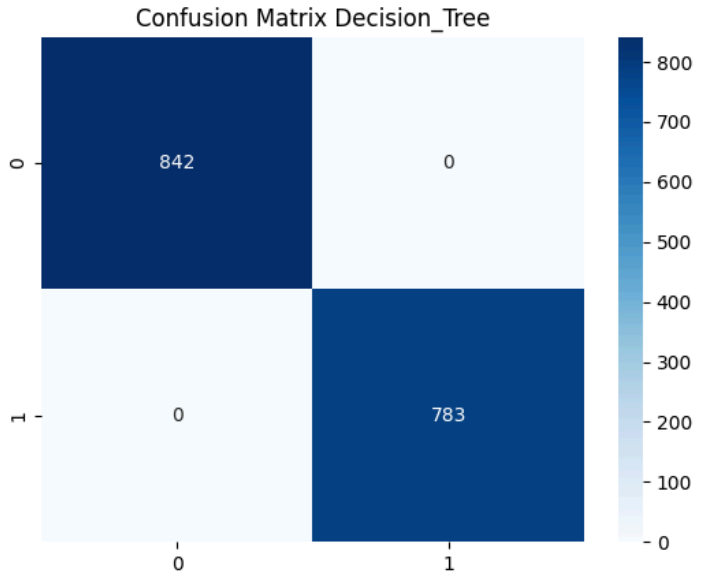


- ROC Curve

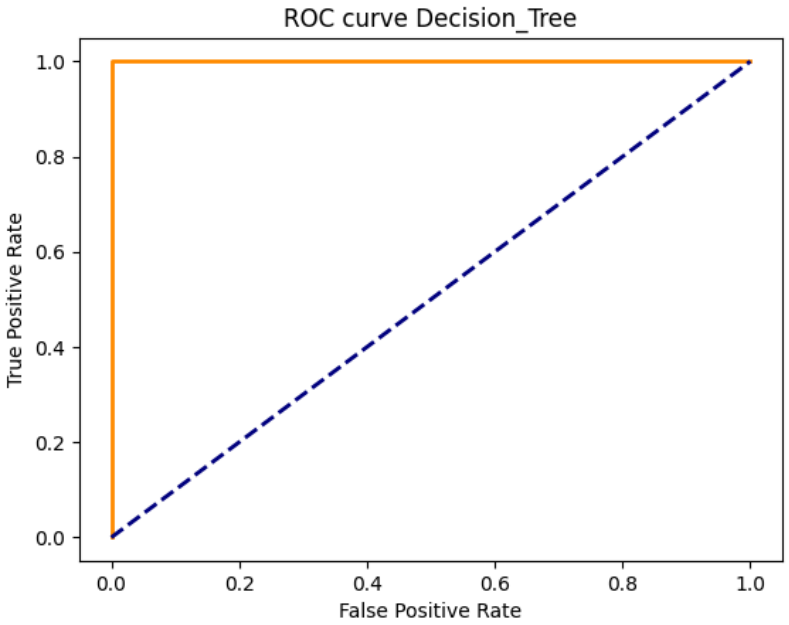


4.2. Decision Tree

- Accuracy: 100%
- Precision: 100%
- Recall: 100%
- F1: 100%
- AUC: 1
- Confusion Matrix

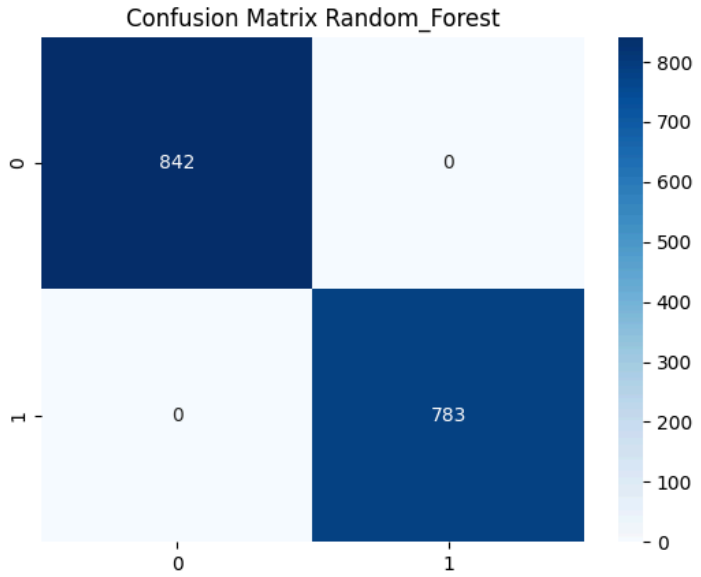


- ROC Curve

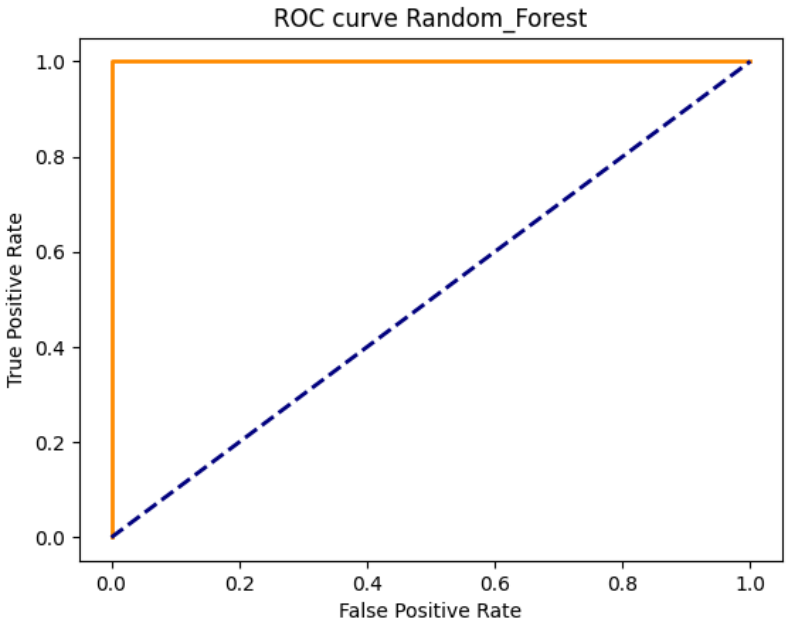


4.3. Random Forest

- Accuracy: 100%
- Precision: 100%
- Recall: 100%
- F1: 100%
- AUC: 1
- Confusion Matrix

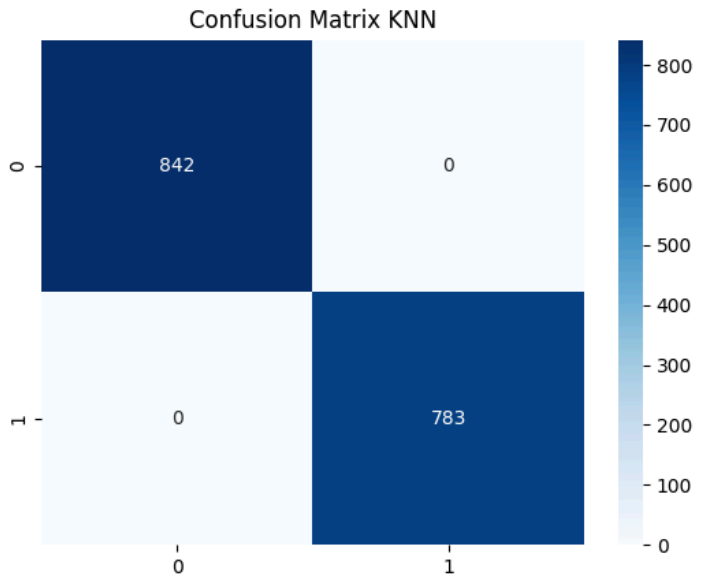


- ROC Curve

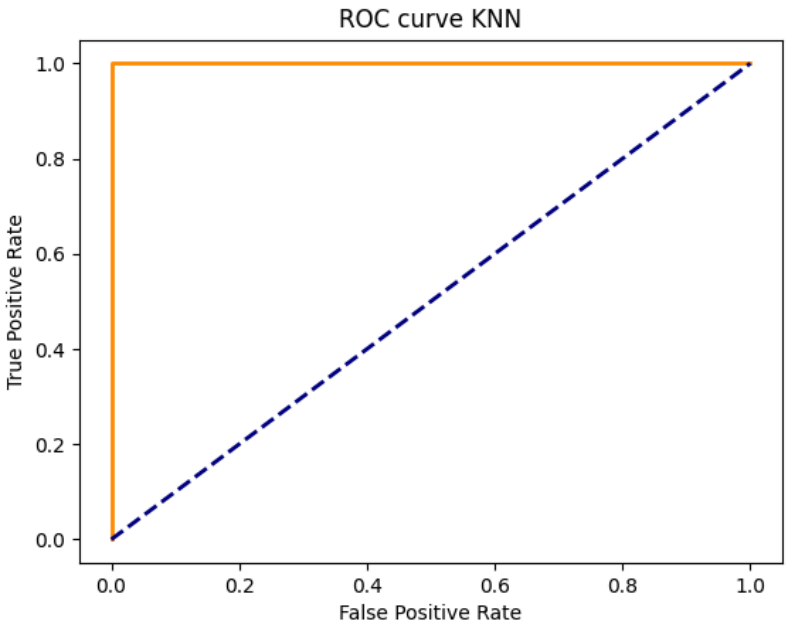


4.4. K-Nearest Neighbors

- Accuracy: 100%
- Precision: 100%
- Recall: 100%
- F1: 100%
- AUC: 1
- Confusion Matrix

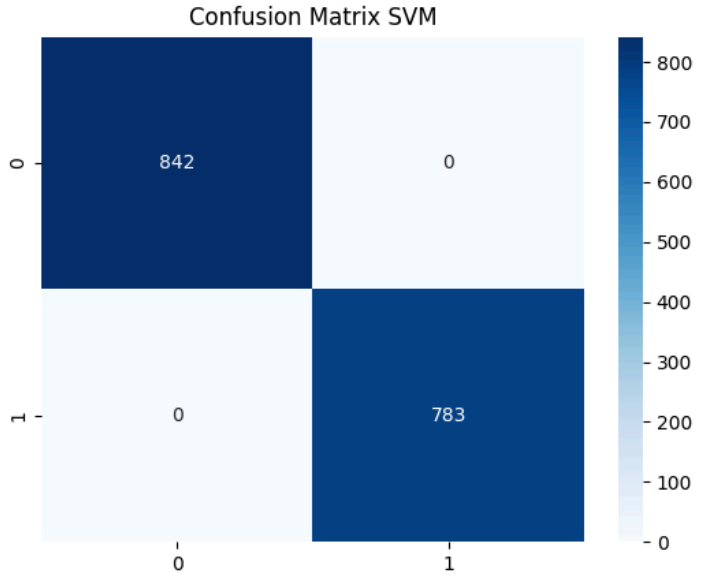


- ROC Curve

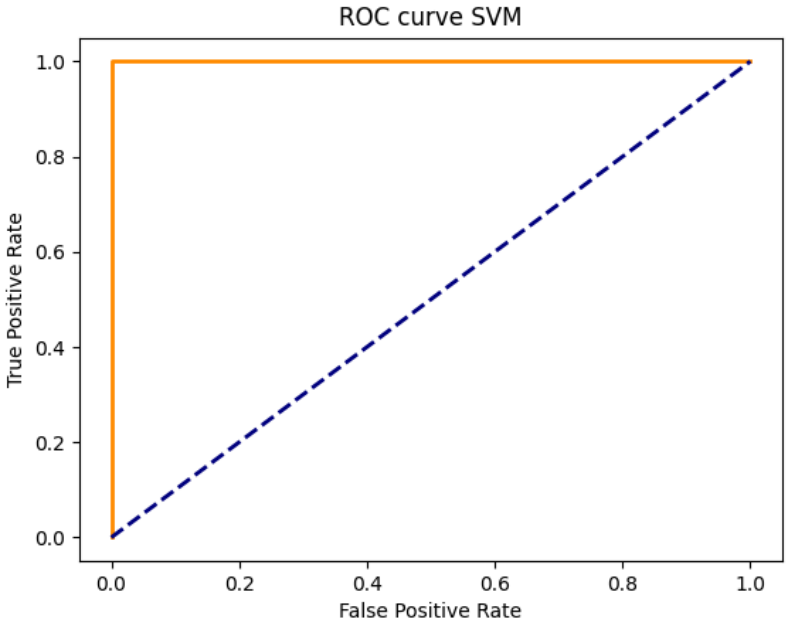


4.5. Support Vector Machines

- Accuracy: 100%
- Precision: 100%
- Recall: 100%
- F1: 100%
- AUC: 1
- Confusion Matrix



- ROC Curve

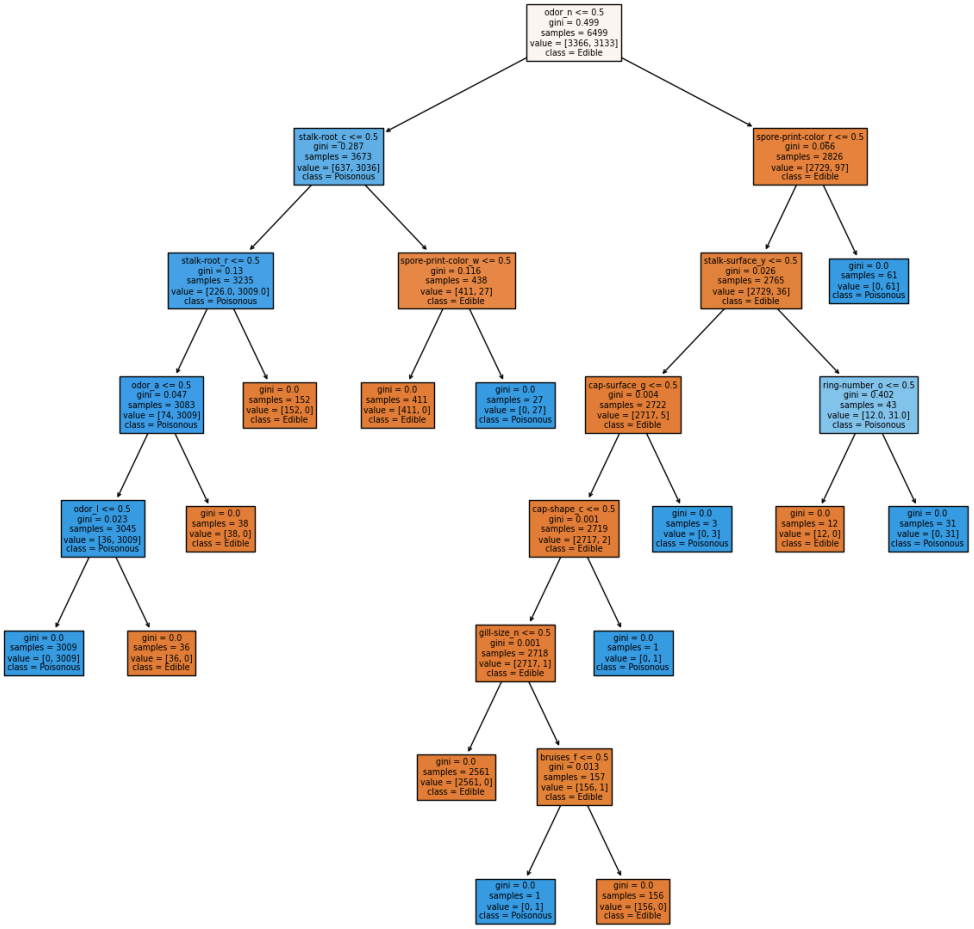


4.6. Comparisons

Come possiamo vedere, tutti i modelli, ad eccezione di Naive Bayes risolvono perfettamente il problema. Potrebbe venir da pensare che i modelli siano in Overfitting, ma poiché non abbiamo duplicati, e il dataset è stato correttamente diviso, possiamo concludere che la qualità dell'inferenza sia ottima.
Questo ci lascia con la domanda: quale modello scegliere? Verrebbe da pensare che la scelta non ha molta importanza, ma possiamo definire un nuovo criterio di valutazione per aiutarci nella scelta: l'explainability. Difatti, tra questi modelli, solo Decision Tree ha un'altissima explainability, in quanto è possibile visualizzare l'albero di decisione prodotto, e quindi codificare una funzione che attraversi i vari nodi di decisione, così da comprendere il perché il modello restituisce un determinato risultato.
Per questa tipologia di problemi (Situazioni che potenzialmente hanno a che vedere con la salute di chi utilizza il modello) poter spiegare all'utente il perché viene visualizzato un determinato output è un enorme vantaggio, in quanto possiamo rassicurarlo, permettendogli di prendere una decisione conscia e magari fare ulteriori ricerche o approfondimenti su quanto predetto.
Per questo motivo il modello che verrà utilizzato è Decision Tree, in particolare implementando una funzione che spieghi l'output in fase di Deployment.

4.7. Decision Tree Graph

Qui di seguito è mostrato l'albero di Decisione prodotto dal modello

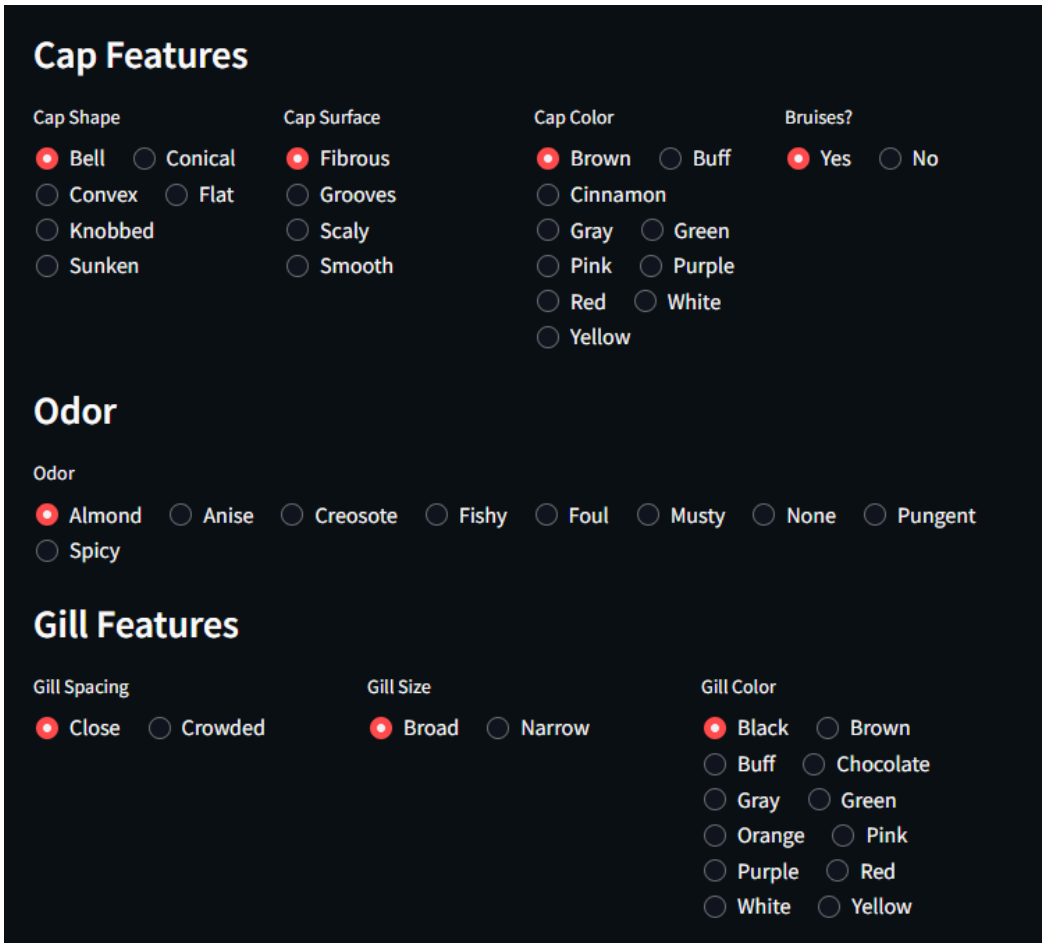


5. Deployment

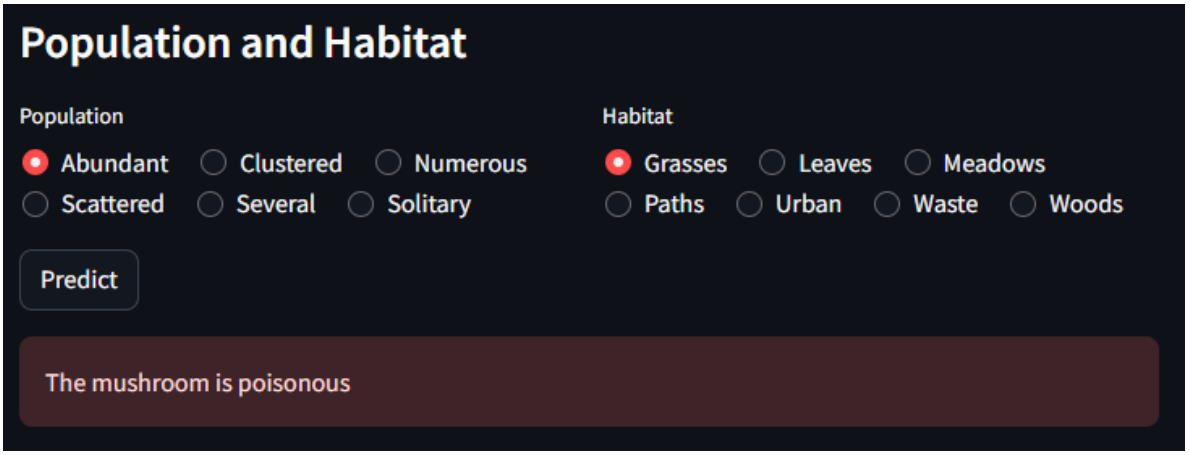
Ora che il nostro modello è pronto all'uso, dobbiamo renderlo utilizzabile, così che possa effettuare previsioni. Per farlo produrremo una semplice web app.

5.1. Web Application

La web app si presenta come una one page application con una serie di radio button che rappresentano le varie feature e i corrispettivi valori. È possibile utilizzarla al link https://fungai.streamlit.app/



Sul fondo è possibile mandare l'input al modello. Alla pressione del tasto verrà poi mostrato l'output con la relativa spiegazione.

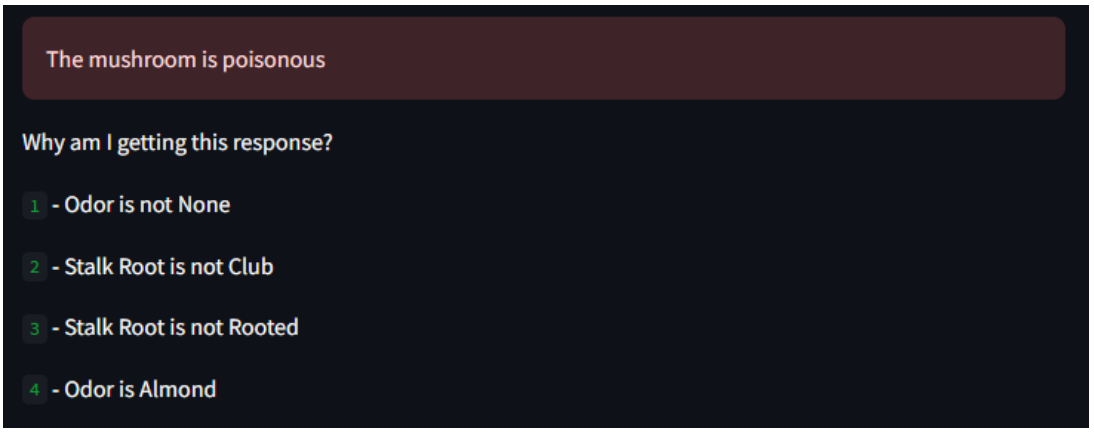


5.2. Explainability

La principale funzionalità della nostra web app deve essere quella di poter spiegare all'utente il perché viene restituito un determinato output. Per farlo è stata implementata la seguente funzione:

```python
def get_decision_explanation(classifier, input_features):
    # Inizializziamo la lista dei risultati
    result = []

    # Dizionario per convertire le feature del One Hot Encoding in feature leggibili dall'utente
    conversion_dict = {...}

    # Lista delle colonne del Dataset
    columns = [...]

    # Cambiamo la forma dell'input in modo che sia compatibile con il modello
    input_features = input_features.reshape(1, -1)

    # Otteniamo il decision path e l'id dei nodi foglia
    node_indicator = classifier.decision_path(input_features)
    leaf_id = classifier.apply(input_features)

    sample_id = 0
    # Ottiene gli id dei nodi attraversati dal campione
    node_index = node_indicator.indices[node_indicator.indptr[sample_id]: node_indicator.indptr[sample_id + 1]]

    for node_id in node_index:
        # Se il nodo è una foglia, non lo consideriamo
        if leaf_id[sample_id] == node_id:
            continue

        # Altrimenti otteniamo il nome della feature e il valore utilizzato per la decisione
        feature_index = classifier.tree_.feature[node_id]
        feature_name = columns[feature_index] if columns else str(feature_index)

        # Aggiungiamo la spiegazione alla lista
        if input_features[sample_id, classifier.tree_.feature[node_id]] == 0:
            verb = "is not"
        else:
            verb = "is"

        explanations = "%s %s %s" % (conversion_dict[feature_name][0], verb, conversion_dict[feature_name][1])
        result.append(explanations)

    return result
```

Quello che fa è, partendo da un input, attraversare l'albero decisionale, e ogni volta salvare il valore del nodo. Dopodiché converte questo in linguaggio naturale, attraverso un dizionario che converte i nomi delle feature nel corrispettivo valore Inglese. Quando viene effettuata una richiesta al modello, verrà visualizzato il seguente output:



In questo specifico caso capiamo che, poiché il rizoma non ha la forma di una clava e non è radicato a terra, e il fungo ha un odore di mandorla, allora è velenoso.

6. Conclusions

In conclusione è possibile dire senza ombra di dubbio che l'obiettivo del progetto è stato raggiunto, arrivando anche a implementare una piccola web app e una funzionalità di explainability.